A Faster Approximate Minimum Steiner Tree Algorithm & Primal-Dual and Local-Ratio Greedy Algorithms

H. Kwon [*] *Jeff Edmonds* [†] *A. Borodin* [‡]

Abstract

We prove matching upper and lower bounds for the Minimum Steiner Tree (MStT) problem within the adaptive priority stack model introduced by Borodin, Cashman, and Magen [2] and lower bounds within the adaptive priority queue model introduced here. These models, which are extensions of the adaptive priority model of Borodin, Nielson, and Rackoff [4], are said to capture reasonable primal-dual and local-ratio extensions of greedy algorithms. Borodin etal. [2] had proved lower bound $1\frac{1}{3}$ on the approximation ratio obtainable for the problem in the stack model. We improve this to $2-\mathcal{O}(\frac{1}{|V|})$. This and similar results prove that the above upper bound is tight, that the adaptive priority stack model is incomparable with the fully adaptive priority branching tree (pBT) model, and that the online stack model is incomparable with the adaptive priority model. We also prove a lower bound in the priority model for complete graphs improving the result of Davis and Impagliazzo [6] from $1\frac{1}{4}$ to $1\frac{1}{3}$. The priority queue model is better because ????? Within it we prove a $\frac{7}{6}$ approximation lower bound.

1 Introduction

We are interested in how well *greedy algorithms* and their *primal-dual* and *local ratio* extensions are able to approximate various variations of the Minimum Steiner Tree Problem. Borodin, Nielsen and Rackoff, [4], introduced the *priority model* which captures reasonable greedy or greedy-like algorithms. The paper of Borodin, Cashman, and Magen [2] formalized a computational model referred to as the *priority stack model* that captures both the primal-dual and the local ratio algorithm design paradigms. The priority queue model is better because ?????

The Minimum Steiner Tree (MST) problem of finding a minimum subtree that spans the terminal nodes has practical applications such as routing VLSI layout, the design of communication networks, and accounting and billing for the use of telephone networks [15]. It is NP-complete.

The best known approximation algorithm [12] for the general MStT problem without any triangle inequality restrictions achieves 1.55 performance guarantee, but it likely fits neither into the priority model nor the stack model. There is a much easier algorithm that achieves a 2-approximation [14]. Simply do a Minimum Spanning Tree on G_R^c, where G_R^c is the graph on the terminal nodes R with weight on edge $\{u, v\}$ being the weight of the minimum path from u to v in G. We show how to express this algorithm within the *adaptive priority stack* model. Given that Davis and Impagliazzo [6] show how to implement Dijkstra single source all destination shortest paths algorithm in the adaptive priority model, it is not hard to see how to do st-connectivity in the adaptive priority stack model. Minimum spanning tree is the classic algorithm in the fixed priority

[*]York University, Canada. james@cs.yorku.ca.
[†]York University, Canada. jeff@cs.yorku.ca. Supported in part by NSERC Canada.
[‡]University of Toronto. bor@cs.toronto.edu.

model. Hence, each step of this MStT algorithm can be implemented by an adaptive priority stack algorithm. However, because this model only allows the algorithm to pass once through the data items, the trick when implementing this algorithm was to prove that it works correctly when all of these tasks are performed simultaneously. Inadvertently, doing so improved the running time of this algorithm from the number $|R|$ of terminal nodes times the running time $\mathcal{O}(|V|\log|V|+|E|)$ of Dijkstra's algorithm to being only the time $\mathcal{O}(|V|\log|V|+|E|)$ of Dijkstra's algorithm once on G.

We also get a 2-approximation algorithm in weaker models when the MStT problem is restricted, namely within the *fixed priority stack* model when there are no edges between the Steiner nodes, within the *online stack* model when there are no edges between the Steiner nodes and the weight of every edge included is one, and within the *fixed priority* model when the triangle inequality holds.

Now let consider lower bounds. When there are only two Steiner nodes s and t, MStT amounts to st-connectivity, which Davis and Impagliazzo [3] proved it cannot be solved by a *fully adaptive priority branching tree* (pBT) algorithm unless it has exponential width. This proves for the first time that this model is incomparable with the adaptive priority model. Previously the adaptive priority stack model had been separated from the much weaker *priority* model for a packing problem (weighted interval scheduling), but this is the first such separation for a problem that is both a graph problem and a covering problem.

When MStT is restricted so that there are no edges between the Steiner nodes, there is no longer a st-connectivity aspect to the MStT problem. Borodin, Cashman, Magen [2] proved a lower bound that no adaptive priority stack algorithm can achieve an approximation ratio better than $1\frac{1}{3}$ in this case even it when restricted to any of the variations listed above. We improve this lower bound to $2-\mathcal{O}(\frac{1}{|V|})$, making it tight. Our lower bound is also in a slightly stronger model in which the algorithm is able to accept *implicit* edges.

We also obtain an arbitrarily large lower bound on the approximation ratio obtained by an adaptive priority algorithm for the MStT problem when it does not need to satisfy the triangle inequality, even when there are no edges between the Steiner nodes are allowed and the weight of every edge included is one. This proves that the online stack model is incomparable with the adaptive priority model. The arbitrarily high lower bound also holds when, though the triangle inequality "holds," and all terminal-terminal edges are implicit and cannot be accepted. This gives the first separation between the initially defined priority model in which implicit edges cannot be accepted and our new strengthening of the model in which they can.

Davis and Impagliazzo Davis [6], before the [2] results, give a lower bound of $1\frac{1}{4}$ within the adaptive priority model when the instance graph to MStT must be complete. We improve this result by increasing the bound to $1\frac{1}{3}$. A weakness of these complete graph lower bounds is that they apply only to graphs of fixed size and hence do not prove that the error is multiplicative instead of additive. We give some intuition as to why proving this same result for arbitrarily large complete graphs would be hard. Proving the same result for in the stack model also seems hard.

Within the priority queue model, we prove a $\frac{7}{6}$ approximation lower bound.

Section 2 provides the preliminary definitions, Section 4 provides the upper bounds, and Section 3 provides the lower bounds.

2 Preliminaries

In this section, we define the *Minimum Steiner Tree* problem (MStT), the *priority model*, and the *priority Stack model*.

	adaptive priority		priority stack		
		br. tree	online	fixed	adaptive
general MStT	∞	∞	∞'	∞'	2
no edges between the Steiner nodes	∞	1	∞'	2	2
& the weight of every edge included is one	∞	1	2	2	2
the triangle inequality holds	2	2	∞'	∞'	2
complete & the triangle inequality holds	$\geq 1\frac{1}{3}$	∞	?		

Table 1: Summary of the known results and results presented in this paper for the MST problem. Here ∞' means that we assume that no constant approximation algorithm possible in the model, but we have not looked for a proof.

A General Computational Problem: An *optimization problem* with priority model $(\mathcal{D}, \Sigma)$ and a family of objection functions $f^n : \mathcal{D}^n \times \Sigma^n \to \mathbb{R}$ is defined as follows. An instance $I = \{D_1, D_2, \ldots, D_n\} \in \mathcal{D}^n$ is specified by a set of n *data items* D_i chosen from the given domain $\mathcal{D}$. A solution for the instance is consists of a decision $a_i \in \Sigma$ made about each data item d_i in the instance I, namely a set of tuples, $S = \{(D_i, a_i) | D_i \in I\}$. An exact algorithm must find a solution that minimizes (maximizes) $f(I, S)$. An $\mathcal{R}$-approximation algorithms finds one within a factor of $\mathcal{R}$ of this optimal value.

The Minimum Steiner Tree (MStT) problem: A data item for the MStT problem in the edge model is a weighted edge $D_i = \langle \{u, v\}, w \rangle$ with the knowledge that the nodes in $N \subseteq V$ are *terminal* nodes and those in $V \setminus N$ are *Steiner* nodes. The decision $a_i \in \Sigma = \{accepted, rejected\}$ to be made is whether to include the edge in the solution. If the resulting solution is a *Steiner tree*, i.e. it spans all the terminal nodes of the instance graph G, then its cost is $f(I, S) = \sum_{a_i = accept} w_i$. If the *triangle inequality* is said to hold then $w(\{u, v\})) \leq w(\{u, x\}) + w(\{x, v\})$.

Adaptive Priority Algorithms: [4] models *greedy* algorithms with the *adaptive priority* model. A *greedy* algorithm receives the data items D_i one at a time and must make an irrevocable decision a_i about each as it arrives. For example, an algorithm for MStT might accept the edge as long as it does not create a cycle. We will assume that the algorithm has unbounded computational power based on what it knows from the data items it has seen already. The algorithm's lack of ability to find an optimal solution arises because it does not know the data items that it has not yet seen. The algorithm, however, is given the additional power to be able specify a priority ordering function (greedy criteria) and is ensured that it will see the data items in this order. For example, an algorithm for MStT may request the cheapest edges between terminal nodes first. [4] allows the algorithm to specify the priority order without allowing it to see the future data items by having it specifying an ordering $\pi \in \mathcal{O}(\mathcal{D})$ of all possible data items. The algorithm then receives the next data item D_i in the actual input according this order. An added benefit of ordering the data items is that when receiving the data item D, the algorithm inadvertently learns that every data item D' before D in the priority ordering is not in the input instance.

algorithm Adaptive Priority Algorithm
$\langle input \rangle$: $I = \{D_1, \ldots, D_n\}$
$\langle output \rangle$: $S = \{(D_i, a_i) |\ 1 \leq i \leq n\}$

begin

Loop: $i = 1, 2, \ldots, n$

$\langle$***loop−invariant***$\rangle$**:** Assume the data items $\langle D_1, D_2, \ldots, D_{i-1}\rangle$ have been seen and processed. This is all that the algorithm knows about the instance.

Choose a possible new ordering Π on all possible data items.

Let D_i be the next data item according to this new order.

Make an irrevocable decision a_{π_i} about the data item D_i.

Update the solution: $S = S \cup \{(D_i, a_{\pi_i})\}$.

end Loop

Output $S = \{(D_i, a_i) |\ 1 \leq i \leq n\}$.

end algorithm

Online, Fixed, and Adaptive Priorities: An adaptive priority algorithm as described above is said to be *adaptive* because it is able to change its priority order Π on the data items each time it sees a new item. In contrast, it is called a *fixed priority* algorithm if it chooses one order that is fixed for the duration of the algorithm, and it is called an *online* algorithm if an adversary chooses the order in which the data items are seen.

Accepting Implicit Edges: Normally, an edge not in the instance graph cannot be included in the solution. In order to gives the algorithm more flexibility, we strengthen the model so they can. An edge is said to be *implicit* if is is not explicitly mentioned in the instance. Its weights is assumed to be that imposed by the triangle inequality, (i.e. the weight of a missing edge $\{u, v\}$ is the length of the shortest path from u to v amongst the included edges). The algorithm can accept or reject an implicit edge any time. This decision however will have to be made without explicitly learning the weight of the edge. Hence, there is no advantage to doing so until after all the explicit edges have been seen. The key thing that such an algorithm cannot do is to initially learn about u and v by saying "Give me the most expensive implicit edge and I will reject it." Note that this addition to the model only makes sense when the triangle inequality holds.

Stack: Borodin, Cashman, and Magen [2] define a new model referred to as the *adaptive stack model*. They advocate that their model captures reasonable primal-dual algorithms and local-ratio algorithms. The weakness of the adaptive priority model is that the algorithm must make an irrevocable decision on the chosen data item, that is, $a_i \in \sum = \{accept, reject\}$. The stack model gives the algorithm a second phase in which it can reject previously "accepted" data items.

The adaptive stack model allows the algorithm to have two phases. The first phase is the same as the adaptive priority algorithms except that the stack algorithm has a stack and a choice set $\Sigma = \{reject,\ stack\}$. The rejected data items are permanently discarded by the algorithm, whereas stacked items are pushed on the top of the stack.

The data items that are placed in the stack are processed during the second phase, which is called the *pop* or *clean up* phase. Each data item is popped in the reverse of the processed order in the first phase. Each time an item is popped, the algorithm *is forced to discard* it if the data items that have been previously been popped and accepted together with those remaining in the stack create a feasible solution. Otherwise, the popped item *is forced to be included* in the algorithm's solution.

Note that the stack model is carefully designed to give the algorithm no control over the second phase. If the algorithm had control over the second phase, then it could read all of the data items during the first phase and push them all onto the stack. Then, knowing the entire input instance, it can use its unlimited power to compute the optimal solution. During the second phase, it could simply accept and reject the data items corresponding to this solution.

Queue: We introduced an new (yet obvious) model referred to as the *adaptive queue model*. It is the same as the previously defined model, except that data items are placed on a queue instead of a stack. For MStT, we can assume that the popping of the second phase does not wait until the pushing phase is completed. In fact, we can assume that the queue aways contains a tree of edges that spans the terminal nodes (once it initially does). Suppose an edge is added to the end of the queue that creates a cycle with what is in the queue. Let e denote the edge that is first in the queue of those in this new cycle. We know that when second phase considers edge e, it will be discarded because what remains in the queue still spans the terminal nodes. Hence, we might as well immediately consider e discarded.

To get some idea of the extra power of having such a queue, suppose we simplify the situation so that instead of the algorithm receiving a sequence individual data items, it receives a sequence of possible solutions. Then there is an online queue algorithm with approximation ratio $c = \sqrt{\frac{f_{max}}{f_{min}}}$, where f_{max} and f_{min} are the maximum and the minimum of any solution value. The value $f_{current}$ of the current solution goes up and down as if the algorithm was walking along a hilly path. If $f_{current}$ ever reaches the value $c \cdot f_{min}$ or smaller, then the algorithm states that it is happy with the solution it has and hence rejects the remaining possible solutions. Even if the adversary can arrange the best possible solution with value f_{min}, the algorithm has achieved its required competitive ratio of c. On the other hand if, all the solutions that appear have value at least $c \cdot f_{min}$, then the algorithm continues until the adversary gets tired and states that all the possible solutions have been seen. The algorithms solution at this time can't be worse than f_{max} and the optimal solution for the instance can't be better than seen by the algorithm and hence not better than $c \cdot f_{min}$. This gives an approximation ratio of $\frac{f_{alg}}{f_{opt}} \leq \frac{f_{max}}{c \cdot f_{min}} = c$. In our lower bound $f_{max} = 8$, $f_{min} = 5$, $c = \sqrt{\frac{f_{max}}{f_{min}}} = 1.265$ and following these same ideas obtained the bounds $\frac{7}{6} = 1.167$. Of course, the "pseudo" upper bound presented above does not work in general, because even though all the individual data items went by, the optimal solution might never have been in the queue at one point in time. Likely, a stronger lower bound could be proved using this fact.

3 Lower Bounds

Section 3.1 describes the general lower bound technique introduced in [4]. Section 3.2 gives the arbitrarily large lower bounds in the priority model. Section 3.3 gives a $\frac{7}{6}$ lower bound in the priority queue model. Section 3.4 gives a $2 - \mathcal{O}(\frac{1}{|V|})$ lower bound in the priority stack model. Finally, Section 3.5 gives the $1\frac{1}{3}$ lower bound in the priority model given a complete graph.

3.1 The Lower Bound Technique

A lower bound in the priority model can be described as a game between an adversary and an algorithm. At the beginning of the ith iteration of the game, the algorithm has been given the partial instance $PI_{i-1} = \langle D_1, D_2, \ldots, D_{i-1} \rangle$ and has committed to the partial solution $PS_{i-1} =$

$\langle a_1, a_2, \ldots, a_{i-1}\rangle$ and the adversary has narrowed down the universe of possible data items to $\mathcal{D}_{i-1}$ from which it promises future data items will chosen. The algorithm being adaptive is allowed to reorder the data items in $\mathcal{D}_{i-1}$. The adversary promises to put in the instance whichever data item in $\mathcal{D}_{i-1}$ is the algorithm's favorite. Hence, the game can be simplified by allowing the algorithm to choose D_i from $\mathcal{D}_{i-1}$. The algorithm then must make an irrevocable decision a_i about D_i. Knowing D_i and a_i, the adversary is then allowed to narrow down $\mathcal{D}_{i-1}$ to $\mathcal{D}_i$ by removing some data items (including D_i) that would make the algorithm's task "easy". Generally, the adversary is allowed to dynamically choose how many data items there will be in the actual instance. Hence, the adversary has the power at any point in the game to declare $I = PI_i$ is the actual instance. The adversary wins if I is a valid instance and $S = PS_i$ is not a sufficiently optimal solution for it.

3.2 Arbitrarily Large Lower Bounds in the Priority Model

Theorem 1. *We obtain an arbitrarily large lower bound on the approximation ratio obtained by an adaptive priority algorithm for the MStT problem when it does not need to satisfy the triangle inequality, even when there are no edges between the Steiner nodes and the weight of every Steiner-terminal edge included is one. The arbitrarily high lower bound also holds when, though the triangle inequality "holds," all terminal-terminal edges are implicit and cannot be accepted.*

This lower bound together with the upper bound in Theorem 8 proves that the online stack model is incomparable with the adaptive priority model. Suppose that in the second scenario, the algorithm was allowed to accept implicit edges. Because all the Steiner-terminal edges have weight one and the triangle inequality holds, the algorithm implicitly knows that the weight on all terminal-terminal edges have weight 2. Hence, it can obtain minimum spanning tree of G_R^c by accepting any spanning tree. Theorem 5 then gives that this is a 2-approximate solution. This gives a separation between the initially defined priority model in which implicit edges cannot be accepted and our new strengthening of the model in which they can be.

Proof of Theorem 1: The instance will have n_t terminal nodes and n_s Steiner nodes. Initially, the universe $\mathcal{D}_0$ of possible data items consists of all the terminal-terminal edges each of weight c, all terminal-Steiner edges each of weight 1, and no Steiner-Steiner edges. Every time the algorithm accepts a Steiner edge, the adversary deletes all other unseen Steiner edges adjacent to this same steiner node. That is, if the algorithm has rejected d edges adjacent to the steiner node s_j and then accepted one adjacent to it, then this node s_j will have degree $d+1$ in the actual instance and the one accepted edge is useless. This continues until either the algorithm has accepted n_t-1 terminal edges or is about to accept a steiner edge adjacent to the last Steiner node considered. At this point, the adversary declares the actual instance consists of all the edges seen, all the terminal edges, and the *star* of steiner edges from this last Steiner node s_{last} considered to each terminal node. The optimal steiner tree, consisting of this star, has weight n_t. The algorithm's total weight is $c(n_t-1)$ if it accepts n_t-1 terminal edges. Otherwise, the process continued until at least one Steiner edge is accepted to each steiner node. Let a denote the number of terminal edges that the algorithms accepts. Because the other Steiner edges go no where, it must accept n_t-a of those to the last steiner node. This gives a total cost of $ca+1\cdot(n_s-1)+1\cdot(n_t-a)$, which when $c \geq 1$ is minimized by setting $a = 0$, giving n_s+n_t-1. Hence, $\mathcal{R} = Alg/Opt = \frac{\min(c(n_t-1),n_s+n_t-1)}{n_t} = \min(c(1-\frac{1}{n_t}), 1+\frac{n_s-1}{n_t})$. This can be made arbitrarily large by making c large (or removing the terminal edges) and by making the number of Steiner nodes much larger than the number of terminal nodes. ■

3.3 A $\frac{7}{6}$-Queue Lower Bound

Theorem 2. *No adaptive queue algorithm achieves better than $\frac{7}{6}$ approximation for the MStT problem. The result still holds even when there are five terminal nodes, all terminal-terminal edges appear and have weight two, the Steiner-terminal edges that appear have weight one, and there are no edges between the Steiner nodes*

Note that if you prefer the size of the solution to be arbitrarily large so that we know that the error is multiplicative instead of additive, than we can repeat this five terminal node lower bound an arbitrary number of times in parallel.

Proof of Theorem 2: As is often the case, the lower bound proof follows the same ideas as the "pseudo" upper bound. As defined for the queue model, at each point in time i, the queue contains a solution, i.e. a tree of edges that spans the five terminal nodes. See Figure 1. The algorithm is able to change this solution by asking for an edge and adding it to the queue. His goal is to obtain a solution of low weight. But as if he were following a hilly path, he might have to go up out of a local minimum before he is able to go down farther. The lower bound adversary allows the algorithm to build at solution of value $f_{current} = cf_{min} = 6.32 \approx 6$. Then the algorithm has a choice. If he says that he is happy with this solution, rejecting all unseen edges, then the adversary can reveal a new Steiner node of degree 5, giving a weight $f_{min} = 5$ solution for a ratio of $\frac{6}{5}$. On the other hand, if the algorithm wants to continue in hopes to find such a weight 5 solution, the adversary can ensure that he must first pass through a weight 7 solution. The adversary then states that there are no more edges in the instance. The algorithm has weight 7, but the optimum is the previous solution of weight 6 for a ratio of $\frac{7}{6}$.

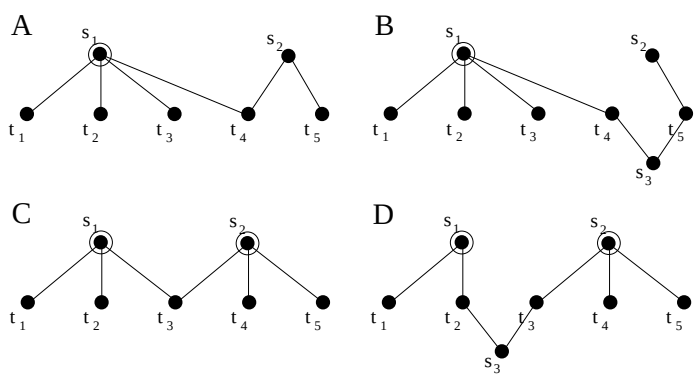

Figure 1: Examples of spanning trees that might be in the queue. A node is circled if it is closed.

To decrease the number of cases to be considered, consider each terminal edge $\{t_k, t_{k'}\}$ as being the pair of Steiner edges $\{s_j, t_k\}$ and $\{s_j, t_{k'}\}$. Then, a useful way of viewing the weight of a tree is the number r of useful Steiner nodes, i.e. those that connect to at least two terminal nodes. Note that if the algorithm were to stop with such a tree, then the unuseful Steiner edges, i.e. those connected with only one terminal node, are automatically removed. A quick check would show that the weight of the resulting solution is $4 + r$. Hence, the goal of the algorithm is to cover the five terminal nodes with a smaller number r of useful Steiner nodes. For example, the weight 6 solutions have $r = 2$ and Figure 1.A and C are the only possibilities.

As defined in the lower bound technique, the adversary also maintains a set $\mathcal{D}_{i-1}$ of the edges that still might get seen. The adversary will assure that each Steiner node s_j is either considered

open in which case all five of its Steiner edges still might be in the input or is considered *closed* in which case all of its not previously seen Steiner edges have been removed from $\mathcal{D}_{i-1}$ and as such are known to not be in the input. Clearly, the adversary will close a Steiner node before the algorithm can see all five of its edges.

The adversary is able to drastically reduce the actions that the algorithm can safely take. If, for example, the algorithm ever considers and rejects an edge $\{s_j, t_k\}$, then s_j must have been open, in which case the adversary can state that all five edges from s_j are in the instance giving an optimal solution of weight 5. The algorithm, no longer having access to this one rejected edge, must have weight at least 6, giving a ratio of at least $\frac{6}{5} > \frac{7}{6}$.

Just like rejected edges, edges popped from the queue are no longer available for the algorithm's solution. Hence, the algorithm best not do this to an open Steiner node's edge. An effect of this is that it is really best for the algorithm to stick with asking for the edges of the same Steiner node until the adversary closes it. In Figure 1.A, for example, suppose the algorithm chooses not to continue asking about Steiner node s_2's edges even though it is still open and instead adds s_3's edges $\{s_3, t_4\}$ and $\{s_3, t_5\}$. Because this creates a cycle, the oldest edge in this cycle, in this case one of s_2's edges, is popped producing Figure 1.B. The adversary can then win by saying that that popped edge was needed in the solution of value 5 stemming from s_2. Similarly, suppose that Figure 1.A had been produced by first adding those edges adjacent to Steiner node s_2, but not closing it moved on to adding those adjacent to s_1. Further suppose that the algorithm then goes back to s_2's edges adding $\{s_2, t_3\}$ to the queue. The effect of this is that edge $\{s_2, t_4\}$ is popped. Again, the adversary can then win by saying that that popped edge was needed.

Now suppose the adversary has allowed the algorithm to build up a solution of weight 6. As said, there are only two such solutions, that in Figure 1.A and that in B. We will now complete the proof by showing that the algorithm cannot continue without increasing the weight to 7. From Figure 1.A, if the algorithm continues with s_2 by pushing edge $\{s_2, t_3\}$, then this creates a cycle popping either $\{s_1, t_3\}$ or $\{s_1, t_4\}$. This produces Figure 1.C. From Figure 1.C, (and similarly from Figure 1.A) if the algorithm asks about other edges then nothing really changes until it has included a Steiner node s_3 with degree of two. In each case, this creates a cycle, popping an edge and increasing the weight to 7. For example, adding edges $\{s_3, t_2\}$ and $\{s_3, t_3\}$ produces Figure 1.D. ■

3.4 A $2 - \mathcal{O}(\frac{1}{|V|})$ Stack Lower Bound

Borodin, Cashman, and Magen [2] define a new model referred to as the *adaptive stack model* and within it prove a $1\frac{1}{3}$ lower bound. We improve this ratio to $2 - \mathcal{O}(\frac{1}{|V|})$. The result is also strengthened by allowing the algorithm to accepted or rejected implicit edges at any time.

Theorem 3. *No adaptive stack algorithm achieves better than $2 - \mathcal{O}(\frac{1}{|V|})$ approximation for the MStT problem. The result still holds even in the following very restricted setting. The triangle inequality holds on the complete graph, though some of the Steiner-terminal edges only exits implicitly and as such can be accepted but not requested. All terminal-terminal edges have weight 2, explicit Steiner-terminal edges have weight 1, implicit Steiner-terminal edges have weight 3, and Steiner-Steiner (which are useless) have weight 2.*

This lower bound is tight (except for the $-\mathcal{O}(\frac{1}{|V|})$) with the adaptive priority stack algorithm for MStT given in Theorem 6. For what it is worth, if you continue to consider the priority model, but restrict it to being either fixed-stack, online-stack, or fixed-no-stack, then Theorems 7, 8, and 9 will show that there is still a 2-approximate algorithm as long as the graphs are restricted in the appropriate ways. Because our lower bounds hold in each of these ways, our result is tight for

each of these restricted models. Specifically, the triangle inequality holds; we could assume that the Steiner-Steiner edges do not exist, because in our lower bound they are effectively useless; the subgraph on the terminal nodes is complete; unless you want all explicit edges to have weight one in which case you can delete these terminal-terminal edges.

Proof of Theorem 3: Our goal is to force the algorithm to stack an expensive solution followed by a cheap solution. When the cheap solution is popped, it is forcefully discarded because there is still a feasible solution, namely the expensive solution, on the stack. Later, on the other hand, the expensive solution will be kept it is required for the solution consisting of what is left on the stack and what has already been accepted. A valid solution for the MStT problem consists of a set of edges that connect the terminal nodes together. Wanting such a solution to be pushed on the stack, the adversary will monitor how well the edges stacked by the algorithm so far connect the terminal nodes. To do this, the adversary partitions the terminal nodes into components that are connected via the stacked edges. Note that $n_t - c$ "edges" are needed to connect the n_t terminal nodes into c components.

Wanting this solution on the stack to be expensive, the adversary wants it to cost the algorithm at least 2 for each "edge" connecting two terminal nodes. There are four types of such connecting "edges". The first type of "edge" is simply a terminal edge $\langle t_k, t_{k'} \rangle$, which as required costs 2. The second type of "edge" consists of two Steiner edges $\langle t_k, s_j \rangle$ and $\langle s_j, t_{k'} \rangle$, which also costs $1 + 1 = 2$. The third type of "edge" needs to be avoided because it is too inexpensive. This consists of an ℓ-star from one Steiner node s_j, fanning out with explicit Steiner edges to some ℓ of the terminal nodes. This acts as $\ell - 1$ connecting "edges" but only costs ℓ, as explicit Steiner edges only cost 1 each. The adversary will avoid allowing the algorithm to push such stars onto the stack by making any unseen edges incident to s_j implicit as soon as the algorithm pushes two explicit edges incident to s_j. The final type of "edge" involves the algorithm accepting an implicit Steiner edge. However, these edges cost 3. We will let r, b, and b' denote the number of accepted "edges" of the first, second, and fourth type. The loop invariant maintains that $r + b + b' \geq n_t - c$ which reconfirms that the number of these "edges" is at least the $n_t - c$ needed to connect the terminal nodes into c connected components. Having enough intuition, we are now ready to prove the theorem.

Before the ith iteration, the set of edges that are still possibly in the instance is denoted $\mathcal{D}_{i-1}$. Initially $\mathcal{D}_0$ contains all terminal and Steiner edges. The adversary announces that a given the Steiner edge $\{t_k, s_j\}$ is implicit by deleting it from this domain $\mathcal{D}_{i-1}$.

We call a stacked edge *necessary* if it does not create a cycle with edges that appear before it on the stack. Otherwise, it is called an *unnecessary* edge. In order to prove the theorem, we first prove that the adversary is able to maintain the following loop invariants.

LI_1: Wrt $\mathcal{D}_{i-1}$, either S_1 or S_2 applies to each Steiner node s_j.

- S_1: At most one necessary explicit Steiner edge incident to s_j has been stacked by the algorithm and all steiner edges incident to it are still in $\mathcal{D}_{i-1}$.
- S_2: Exactly two necessary explicit Steiner edges are incident to s_j and each of the rest of the edges incident to s_j has been rejected, stacked as unnecessary edge, or removed by the adversary causing it to be implicit.

LI_2: A relation $r + b + b' \geq n_t - c$ holds.

We prove the loop invariants are maintained as follows. Initially, no edge has been seen. Therefore, S_1 applies to all Steiner nodes, and $r = b = b' = 0$ and $c = n_t$, which results in $r + b + b' \geq n_t - c$. Therefore, the base case clearly holds. Now assume that the loop invariants hold at the beginning

of ith iteration. The Figure 2 depicts a possible adversarial set $\mathcal{D}_{i-1}$. Imagine that an algorithm considers an edge. No matter what type it is, nothing changes if it is rejected by the algorithm. Hence, we consider only acceptance cases.

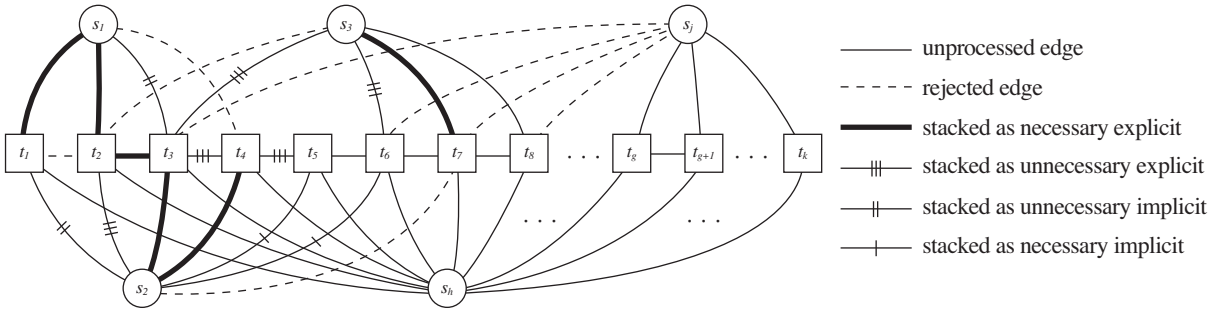

Figure 2: A possible situation at the beginning of the ith iteration.

Case 1: The algorithm stacks a necessary terminal edge $D_i = \{t_k, t'_k\}$. The adversary, in turn, does nothing, yielding $\mathcal{D}_i = \mathcal{D}_{i-1}$. Since D_i is a terminal edge, it does not affect S_1 or S_2. The c decreases by one, r increases by one, and b and b' are unchanged. Hence, the relation $r+b+b' \geq n_t - c$ still holds.

Case 2: The algorithm stacks an explicit Steiner edge $D_i = \{t_g, s_j\}$, which is the first stacked edge incident to s_j. The adversary does nothing, yielding $\mathcal{D}_i = \mathcal{D}_{i-1}$. The s_j must have been of type S_1 and remains to be type S_1. Hence b, b', r, c, and $r+b+b' \geq n_t - c$ are unchanged.

Case 3: The algorithm stacks the second necessary and explicit Steiner edge $D_i = \{t_k, s_j\}$ incident to Steiner node s_j. The adversary, in turn, removes the rest of the unseen edges incident to s_j, yielding $\mathcal{D}_i \subset \mathcal{D}_{i-1}$. These removed edges become implicit edges. Moreover, S_2 now applies to s_j instead of S_1, increasing b by one. Since D_i is necessary, it does not create a cycle with previously stacked edges and hence c decreases by 1. The r and b' are unchanged. Hence, the relation $r+b+b' \geq n_t - c$ is true.

Case 4: The algorithm considers an edge which creates a cycle with stacked edges. By definition, such an edge is called unnecessary. Whether the algorithm decide to stack or to reject it, the adversary does nothing, yielding $\mathcal{D}_i = \mathcal{D}_{i-1}$. Note b and c does not change, but r or b' may increase by one. Hence, the relation $r+b+b' \geq n_t - c$ still holds.

Case 5: The algorithm stacks a necessary and implicit Steiner edge $\{t_{k'}, s_j\}$ without actually seeing it. If this edge is still in $\mathcal{D}_{i-1}$ then the adversary inform the algorithm that actually this edge is explicit in the actual input instance. It is just that the algorithm has not yet read this part of the input. This case can be handled by Case 2, 3, or 4. If the edge being accepted is not in $\mathcal{D}_{i-1}$, the adversary informs the algorithm that this is in fact an implicit edge and its weight is 3. In latter case, the adversary in turn does nothing, yielding $\mathcal{D}_i = \mathcal{D}_{i-1}$. Since the edge is implicit, it does not affect S_1 nodes. The c decreases by one, b' increases by one, and r and b are unchanged. Hence, the relation is still true.

Based on the five cases above, we conclude that the loop invariants are maintained. When c becomes 1, the adversary knows that the terminal nodes are connected by the stacked edges into one component. Hence, he knows that there is an expensive solution pushed on the stack. The

adversary at this point stops this first phase of the game. He declares to the algorithm that the actual input instance consists of the edges seen by the algorithm so far together with all the edges remaining in $\mathcal{D}_i$. The next task is to prove that this instance contains a cheap optimal solution. The first step to accomplish this is to note that at most n_t-1 Steiner nodes of type S_2 connect the n_t terminal nodes into one component. Because there are $n_s \geq n_t$ Steiner nodes, the loop invariant ensures us that there is at least one Steiner node remaining of type S_1. All the edges incident to such a node have not ever been deleted from $\mathcal{D}_i$ and hence all appear explicitly in the input. This forms a n_t-star which in itself is an optimal solution of weight n_t.

At the end of this first stage of the game, the algorithm still must make decisions about the remaining data items in the input instance (which according to the adversary will be those edges remaining in $\mathcal{D}_i$). The algorithm may decide to stack them or decide to reject them. It does not actually matter to the adversary, because either way these edges will be rejected at the very beginning of the pop stage of the algorithm. The reason for this is that the adversary was careful that there is an expensive solution previously pushed on the stack and hence these remaining edges in $\mathcal{D}_i$ are not needed.

As the rest of the pop stage of the algorithm proceeds, we need to prove which of the algorithm's stacked edges get forcefully accepted and which get forcefully rejected. This happens in reverse of the pushed order. Any edge labeled as unnecessary will be rejected when it is popped. It formed a cycle with the edges already pushed on the stack when it got pushed on and hence it will form the same cycle with the edges still on the stack when it is popped. The one necessary Steiner edge incident to a Steiner node of type S_1 will also be rejected. All other edges incident to this Steiner node either was initially rejected and not stacked or has already been popped and rejected by the pop stage. Hence, this edge is not a part of a remaining solution. What will get forcefully accepted when popped will be the "edges" forming the expensive solution. As they got stacked by the algorithm these were needed to merge the connected components of the terminal nodes together and hence are all needed for the solution. Each terminal edge has weight 2. There are two Steiner edges incident to each Steiner node of type S_2, each with weight 1. Each implicit Steiner edges incident has weight 3. Hence the cost of this popped and accepted solution is $2 \cdot r + (1+1) \cdot b + 3 \cdot b' \geq 2(r+b+b')$ which by the loop invariant is at least $2(n_t - c)$ and by the termination condition is $2(n_t - 1)$. Above we showed how the cost of the optimal solution is only n_t. Hence, we can summarize the competitive ratio to be $\mathcal{R} = \frac{ALG}{OPT} \geq \frac{2(n_t-1)}{n_t} = 2 - \frac{2}{n_t} = 2 - \mathcal{O}(\frac{1}{|V|})$. ■

3.5 A $1\frac{1}{3}$ Priority Lower Bound given a Complete Graph

Davis and Impagliazzo [6] proved the first priority lower bound for the MStT problem, where the approximation ratio is $1\frac{1}{4}$. This result is still incomparable with the lower bounds of 2 given above because they give their results when the input graph must be a complete *quasi bipartite* graph, meaning that there are no edges between the Steiner nodes, but all other edges are explicitly in the instance. Playing with their cases further, we managed to improve their lower bound to $1\frac{1}{3}$.

Imposing that the instance must be a complete quasi bipartite graph makes proving a lower bound harder. The adversary can no longer simple delete an edge $\langle u, v \rangle$, it must either delete all the edges incident to the node u or all those incident to v. Although such a step $\mathcal{D}_i \subseteq \mathcal{D}_{i-1}$ is really a reduction in the set of edges, we call this *the adversary deleting the node u (or v).*

Theorem 4. *No adaptive priority algorithm can achieve better than $1\frac{1}{3}$ approximation for the MStT problem even when restricted to complete quasi bipartite graphs in which the triangle inequality holds.*

Proof of Theorem 4: The adversary initially constructs a set $\mathcal{D}_0$ shown in Figure 3.

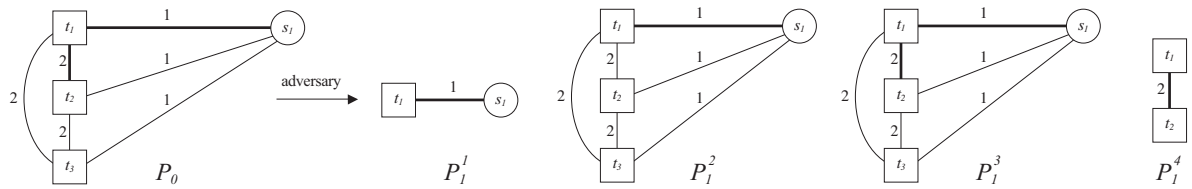

Figure 3: The adversarial set $\mathcal{D}_0$ and a bold edge is assumed to be considered first by the algorithm. Each right of $\mathcal{D}_0$ is the second adversarial set $\mathcal{D}_1 \subseteq \mathcal{D}_0$ reduced by the adversary. The squared nodes are terminals and the circled ones are Steiner.

Case 1: If the algorithm initially considers a Steiner edge, by symmetry we can assume that it considers $D_1 = \{t_1, s_1\}$. If the algorithm accepts the edge, the adversary declares that the actual instance is this one edge, denoted P_0^1. Note, it is a complete graph as promised. Accepting this edge was a mistake, giving the approximation ratio is $\mathcal{R} = \frac{ALG}{OPT} = \frac{1}{0} \approx \infty$. If the algorithm rejects this edge, the adversary gives the entire graph as the instance, denoted P_1^2. No matter what future decisions are made, the best cost for the algorithm is 4 whereas the optimal cost is by taking the three Steiner edges. Therefore, the approximation ratio is $\mathcal{R} = 1\frac{1}{3}$.

Case 2: If the algorithm initially considers a terminal edge, by symmetry we can assume that it considers $D_1 = \{t_1, t_2\}$. If the algorithm accepts the edge, again the adversary gives the entire graph, P_1^3 again giving $\mathcal{R} = 1\frac{1}{3}$. If the algorithm rejects the edge, the adversary removes all the remaining edges, shown as P_1^4. In this case, the algorithm even fails to find a valid Steiner tree.

An arbitrary large lower bound can be obtained by having much bigger weights on the terminal edges. This, however, violates the triangle inequality. ■

A concern with our above $1\frac{1}{3}$ result and Davis and Impagliazzo [6]'s $1\frac{1}{4}$ result is that only small graphs, instances with at most four nodes, are considered. Because the lower bound does not consider arbitrarily large graphs, it does not prove that this error is multiplicative.

One feature of our $1\frac{1}{4}$ and their $1\frac{1}{3}$ result is that it holds even when the algorithm knows a priori not necessarily which nodes and hence which edges are present in the instance, but what the weight of each edge will be if it is present. Though this does strengthen the result slightly, because it gives the algorithm more power and the adversary less, the true motivation for this change is that it significantly simplifies the proof. When arbitrary weights are possible and the algorithm sorts the data items according to these weights in an arbitrary way, then it is hard for the adversary to keep track of which weights are still possible.

This motivates us to want to prove a lower bound with fixed weights, arbitrarily large n, and a complete graph. It seems, however, that this gives too much power to the algorithm. We now outline a strange and unfair algorithm that uses tricks to learn the input instance and then uses its unbounded computational power to obtain what we conjecture to be a near optimal solution.

Conjecture 1. *A priority algorithm can achieve a $1 + O(\frac{1}{k})$ upper bound for the MStT problem with the weights on each possible edge is known a priori but which nodes in the graph are not.*

Proof Sketch for Conjecture 1 Because the weights are fixed and the graph is complete on the nodes that are in the graph, to learn the entire instance, the algorithm need only learn which nodes are

in the instance. To learn this, the algorithm need only to see one edge incident to each node. An algorithm can do this by asking for and reject the most expensive edge incident to each node. One has to be a little careful the one does not disconnect the graph. But that aside, we conjecture that there exists an nearly optimal solution not containing any of these rejected edges. The algorithm uses its unbounded computational power to find such a near optimal solution. ■

4 Upper Bounds

This section provides the upper bound for four variations of the Minimum Steiner Tree Problem on four variations of the priority model. We start by considering the most general version of the problem, i.e. there may be edges between the Steiner nodes and the triangle inequality may not hold. Takahashi and Matsuyama [13] and Kou, Markowsky and Berman [9] give a well known 2-approximation algorithm. Given a graph G, let G_R^c denote the metric completion of G on the terminal nodes, namely for every pair of terminal nodes $u, v \in R$, the cost of edge $\{u, v\}$ in G_R^c is the length of the shortest path from u to v in G. Let $MST(G_R^c)$ denote a Minimum Span Tree of G_R^c. Let $Alg(G)$ denote the union over all terminal edges $\{u, v\} \in MST(G_R^c)$ of a shortest path in G from u to v (if necessary remove edges in order to break cycles). The weight of this Steiner tree is at most twice the optimal Steiner $MStT(G)$. See Figure 4 for an instance for which there is a factor of two loss. The optimal has weight $|R|$ consisting of the star branching from the single Steiner node to each terminal node. The solution $Alg(G)$ has weight $2(|R| - 1)$ consisting of the path through the terminal nodes. Figure 5 gives a more complex example. Part of the metric completion G_R^c of G is given on the bottom right. Its minimal spanning tree is in bold. The solution $Alg(G)$ consists of every edge in G except for $\{a, b\}$. ($\{b, v\}$ is not bold in order to demonstrate something else. This solution is in fact optimal.

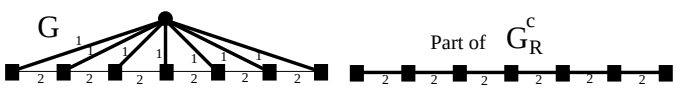

Figure 4: An instance for which there is a factor of two loss.

Theorem 5. $Alg(G) \leq 2MStT(G)$

Proof of Theorem 5: Let T be a Minimum Steiner Tree of the instance graph G, i.e. spanning all the terminal nodes R. From this, form a traveling salesman tour, TST, in G that visits every node in R by embedding the tree T as a wall in the plane, placing ones left hand on the wall, and tracing the outside of it until you return. Because each edge is visited twice, $w(TST) = 2w(MStT(G))$. Figure 5 shows an example of T in bold, TST in curve, and SG in a dotted line. Here SG is a spanning graph of G_R^c formed from TST by including edge $\{u, v\}$ if v is the next terminal node visited after u in this tour. Because the weight on G^c edges are the length of the shortest paths in G, we have that $w(SG) \leq w(TST)$. From this, form a spanning tree ST of G_R^c by deleting enough edges to break all cycles. This gives, $w(MST(G_R^c)) \leq w(ST) \leq w(SG)$. The algorithm produces a Steiner tree $Alg(G)$ such that $w(Alg(G)) \leq w(MST(G_R^c))$. ■

The obvious way to find the metric completion G_R^c is to perform Dijkstra's algorithm rooted at each of the terminal nodes, hence requiring time $|R| \times \mathcal{O}(|V| \log |V| + |E|)$.[1] Mehlhorn [11] finds a

[1] Dijkstra's algorithm can be sped up from $\mathcal{O}(|E| \log |V|)$ to $\mathcal{O}(|V| \log |V| + |E|)$ by using a *fractal* data base [**?**] instead of a heap for the priority queue.

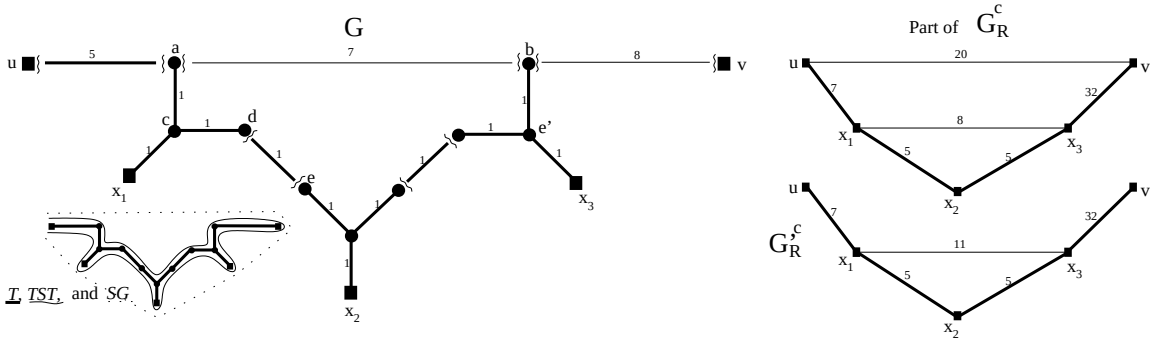

Figure 5: An instance G. The key edges in the metric completion G_R^c with its minimum spanning tree in bold is used to demonstrate the 2-approximate minimal Steiner tree algorithm. The resulting Steiner tree in G (which is optimal) consists of the bold edges plus $\{b, v\}$. T, TST, and SG used in the proof of correctness of this algorithm is also given. The graph G is partitioned into $N(u)$ and $G_R'^c$ with its minimum spanning tree in bold is given as done in the Mehlhorn algorithm. The edges bold in G are those pushed on the stack by our algorithm when $\ell = 10$ just before the edge $\{a, b\}$ is considered. It is used to demonstrate the proof of correctness of our algorithm.

$\mathcal{O}(|V| \log |V| + |E|)$ implementation of this algorithm by performing one Dijkstra's algorithm rooted at an axillary node s_0 connected to each of the terminal nodes. This is done in order to partition the nodes into $\{N(u) \mid u \in R\}$, where $N(u)$ is the set of nodes which are closer to the terminal node u than to any other terminal node. Figure 5 shows an example instance G partitioned into these neighborhoods. Mehlhorn then does a second pass through all the edges of the graph in order to form a graph $G_R'^c$ consisting of pairs of terminal nodes $\{u, v\}$ that are close enough so that their neighborhoods $N(u)$ and $N(v)$ are connected by a single edge $\{a, b\}$. The weight of an edge $\{u, v\}$ in $G_R'^c$ is the weight of the minimum path between u and v that includes only nodes from $N(u)$ and $N(v)$. The figure shows the corresponding graph $G_R'^c$. Note that the weight 19 of $\{x_1, x_3\}$ is larger than the weight 8 of the minimum x_1x_3-path. Despite these differences between $G_R'^c$ and G_R^c, they prove that a minimum spanning tree of $G_R'^c$ is also a minimum spanning tree of G_R^c. More over, they proof that for each edge $\{u, v\}$ in $MST(G_R'^c)$, the found path between u and v in G is of minimum weight. Hence, Mehlhorn can complete the algorithm by finding $MST(G_R'^c)$ and returning these paths.

In order to fit their algorithm into the adaptive priority stack model, we need to change their implementation in a few ways. They find the edge $\{a, b\}$ connecting neighborhoods $N(u)$ and $N(v)$ during a second pass. We are not allowed such a second pass. Some such edge $\{a, b\}$ can easily be found during the first pass as the neighborhoods $N(u)$ and $N(v)$ merge. However, we show how to slightly modify the Dijkstra priority on the edges so that the connecting edge $\{a, b\}$ first found is one that gives a shortest path from u to v. Unlike them, we are not able to compute the minimum spanning tree of $G_R'^c$ in a separate phase. Our modification on $\{a, b\}$'s priority also ensures that the pairs of terminal nodes $\{u, v\}$ will be connected in the order mirroring the order that edges $\{u, v\}$ are added to $MST(G_R^c)$ by the minimum weight greedy algorithm. Their algorithm needs a final stage that collects together the shortest uv-paths for each edges $\{u, v\} \in MST(G_R'^c)$ and breaks any cycles. Instead, our algorithm pushes on the stack all the edges in the shortest paths trees expanding out from the terminal nodes. We prove that the pop phase defined for the stack model automatically deletes all unnecessary edges leaving only the edges in $Alg(G)$.

Theorem 6. *There is a 2-approximate adaptive priority stack algorithm solving the MStT problem*

even when there may be edges between the Steiner nodes and the triangle inequality may does not hold.

This upper bound together with the lower bound in [3] for st-connectivity proves that the adaptive priority stack model is incomparable with the fully adaptive priority branching tree (pBT) model.

Proof of Theorem 6: Because our algorithm wants to find a short path between every pair of terminal nodes u-v, it uses this Dijkstra's algorithm to simultaneously expand a shortest paths tree around each terminal node $u \in R$. It is easier to visualize the synchronizing of these expansions by as having a parameter ℓ slowly increase. For each edge $\{a, b\}$ and terminal node u, define $\ell_{\langle\{a,b\},u\rangle}$ to be the length of the shortest path from u that includes the edge $\{a, b\}$. When ℓ reaches this value, this edge will be added to the tree expanding out of u, assuming that it has not already been added elsewhere. Similarly, define $\ell_{\langle\{a,b\},\{u,v\}\rangle}$ to be half the length of the shortest path from u to v, if the edge $\{a, b\}$ contains this half way point. When ℓ reaches this value, this edge, if not already used, will be used to merge the tree expanding from u and that expanding from v. For example, in Figure 5 when $\ell = \frac{5}{2}$, the tree expanding from terminal node x_1 merges with that expanding from x_2 by having the edge $\{d, e\}$ added. At this same moment, the edge $\{x_1, x_2\}$ is added to the minimal spanning tree of the metric completion graph G^c_R. Define $\ell_{\{a,b\}} = \min(\min_{u\in R} \ell_{\langle\{a,b\},u\rangle}, \min_{u,v\in R} \ell_{\langle\{a,b\},\{u,v\}\rangle})$ to be the value for the parameter ℓ at which the edge $\{a, b\}$ is first considered in one of these ways.

This is implement in the adaptive priority stack model simply by choosing next the unseen edge with the smallest $\ell'_{\{a,b\}}$ value. Here $\ell'_{\{a,b\}}$ is $\ell_{\{a,b\}}$ if all the other edges in that path up to edge $\{a, b\}$ have already been seen so that the algorithm can know the value $\ell_{\{a,b\}}$.

When the edge $\{a, b\}$ is considered, it is pushed onto the stack if it does not creates a cycle with the previously pushed edges, otherwise, it is rejected. Let $MST(G^c_R)$ denote the minimal spanning tree of G^c_R, where ties are broken in the same way as done in this algorithm. Lemma 1 proves that for each edge $\{u, v\} \in MST(G^c_R)$, the edges pushed on the stack by our algorithm contains a shortest path from u to v, i.e. a path with weight equal to that of edge $\{u, v\} \in G^c_R$. Because the edges $\{u, v\} \in MST(G^c_R)$ span the terminal nodes, so does the union of these shortest paths from u to v. Being a tree, the edges in this union and no edges outside it are kept when the stack pops, i.e. the pushed edges connecting the unnecessary Steiner nodes are deleted. Hence, the weight of the Steiner tree returned by this algorithm is at most that of $MST(G^c_R)$, which by Theorem 5 is at most twice the optimal $MStT(G)$. ■

Lemma 1. *Let $MST(G^c_R)$ denote the minimal spanning tree of G^c_R, returned by the standard minimum weight greedy algorithm where ties are broken in the same way as done in the Theorem 6 algorithm. For each edge $\{u, v\} \in MST(G^c_R)$, the edges pushed on the stack by contains a shortest path from u to v. For example, this true for $\{x_1, x_2\}$ in Figure 5 but not for for $\{u, v\}$.*

Proof of Lemma 1: We prove the contra-positive. Denoted by $P_{\{u,v\}}$ the shortest path between terminal nodes u and v that would be found by our algorithm if these were the only nodes from which shortest paths trees expanded. Suppose there is an edge $\{a, b\} \in P_{\{u,v\}}$ that is not pushed on the stack. We will prove that the when the $MST(G^c_R)$ algorithm considers the edge $\{u, v\}$, it is not accepted. Figure 5 was designed in particular to help with this proof.

The reason edge $\{a, b\}$ is not pushed on the stack when it was examined is because it created a cycle with already pushed edges. Let $P_{\{a,b\}}$ denote this pushed ab-path. Label each edge $\{c, d\}$ in $P_{\{a,b\}}$ with the terminal node x_i at the root of the shortest paths tree that added the edge. If $\{c, d\}$ merged trees rooted at x_i and x_{i+1} then give the edge either label. Because these label appear contiguously, we can let $\langle x_1, \ldots, x_r\rangle$ denote the sequence of distinct terminal nodes labeling the edges in $P_{\{a,b\}}$. Let P^c denote the uv-path $\langle u, x_1, \ldots, x_r, v\rangle$ in G^c_R. For each $\{x_i, x_{i+1}\} \in P^c$, we will

show that $w^c_{\{x_i,x_{i+1}\}} \leq w^c_{\{u,v\}}$, giving that the $MST(G^c_R)$ algorithm considers the edge $\{x_i, x_{i+1}\}$ before $\{u, v\}$. This ensures that when the $MST(G^c_R)$ algorithm considers the edge $\{u, v\}$, it does not accept it because it forms a cycle with the edges that it has already accepted. This completes the contra-positive proof of the lemma.

Let $\{c, d\}$ and $\{d, e\}$ be consecutive edges $P_{\{a,b\}}$ that are labeled respectively with x_i and x_{i+1}. Note that $w^c_{\{x_i,x_{i+1}\}} \leq w(P_{\{x_i,d\}}) + w(P_{\{x_{i+1},d\}})$ because by definition $w^c_{\{x_i,x_{i+1}\}}$ is the weight of the shortest path from x_i to x_{i+1} in G and $P_{\{x_i,d\}} + P_{\{x_{i+1},d\}})$ is such a path. By the label of x_i on the edge $\{c, d\}$, we know that either $\{c, d\}$ was pushed to be a part of the path from x_i to d in the shortest paths tree rooted at x_i or was pushed to extend this path. Either way, we have that $w(P_{\{x_i,d\}}) \leq \ell_{\{c,d\}}$. Next we have $\ell_{\{c,d\}} \leq \ell_{\{a,b\}}$ because the edge $\{c, d\}$ was pushed by our $MStT(G)$ algorithm before $\{a, b\}$ was. We have that $\ell_{\{a,b\}} = \min(\min_{u' \in R} \ell_{\langle\{a,b\},u'\rangle}, \min_{u',v' \in R} \ell_{\langle\{a,b\},\{u',v;\}\rangle}) \leq \ell_{\langle\{a,b\},\{u,v\}\rangle} = \frac{1}{2}w(P_{\{u,v\}}) = \frac{1}{2}w^c_{\{u,v\}}$. Similarly, $P_{\{x_{i+1},d\}}) \leq \frac{1}{2}w^c_{\{u,v\}}$, completing the result that $w^c_{\{x_i,x_{i+1}\}} \leq w^c_{\{u,v\}}$ and in doing so completing the proof. ■

Though [11] has already implemented their MStT algorithm to run in time $\mathcal{O}(|V| \log |V| + |E|)$, it is interesting to see the adaptive priority stack model can also do it in this time.

Lemma 2. *As stated, our implementation of the algorithm requires time $\mathcal{O}(|E| \log |E|)$, but another small change will decrease the time back to $\mathcal{O}(|V| \log |V| + |E|)$.*

Proof of Lemma 2: Dijkstra's algorithm can be sped up from $\mathcal{O}(|E| \log |V|)$ to $\mathcal{O}(|V| \log |V| + |E|)$ by using a *fractal* data base [**?**] instead of a heap for the priority queue. Removing the minimum data item still takes $\mathcal{O}(\log |V|)$ time but decreasing the priority value ℓ of an item now only takes $\mathcal{O}(1)$ amortized time. Dijkstra, using node data items, removes the minimum item $|V|$ times and decreases a priority $2|E|$ times, giving the stated times. Theorem 6, using edge data items, removes the minimum item $|E|$ times, giving the old Dijkstra time $\mathcal{O}(|E| \log |E|)$. To get the improved time, the adaptive priority stack model needs to have both node data items and edge data items. The node data items contain pointers to the adjacent edges and the edge data items contain pointers to the adjacent nodes. The node data items are taken as the trees grow. The data item $\{a, b\}$ is taken only as the tree growing from terminal node u to node a merges with that growing from v to b. ■

In order to better understand both the minimum Steiner tree problem and variations of the priority model, we will now consider upper bounds for three restrictions of the problem on three more variations of model. The first restriction requires that there are no edges between the Steiner nodes, i.e. the graph is *quasi bipartite*. The following algorithm is surprisingly simple and does not require reordering the edges each iteration.

Theorem 7. *There is a 2-approximate* **fixed stack** *priority algorithm solving the MStT problem when there are no edges between the Steiner nodes, even when the triangle inequality does not hold.*

Proof of Theorem 7: The push phase of the stack algorithm pushes a minimum spanning tree, $MST(G)$, of the entire graph G. If a Steiner node only has one adjacent edge pushed, then this edge is not necessary for spanning the terminal nodes and this edge is deleted in the pop phase of the algorithm. Let Alg denote the popped and accepted edges. Let S denote the set of Steiner nodes that are used, i.e. those with degree at least two in Alg, and $\overline{S}$ the remaining Steiner nodes. For each Steiner node s, let e_s denote the cheapest edge adjacent to s and w_s its weight. Note this edge goes to a terminal node because there are no edges between the Steiner nodes. Because each $s \in S$ has degree at least two in Alg, we have that $2\sum_{s \in S} w_s \leq w(Alg)$. The algorithm first pushes $Alg \cup \bigcup_{s \in \overline{S}} e_s$ and hence this is a minimum spanning tree of G. $MStT(G)$ spans the terminal nodes

and hence $MStT(G) \cup \bigcup_{s \in S \cup \overline{S}} e_s$ spans all of G. This gives that $w(Alg) + \sum_{s \in \overline{S}} w_s \leq w(MStT(G)) + \sum_{s \in S \cup \overline{S}} w_s$ or simplified to $w(Alg) \leq w(MStT(G)) + \sum_{s \in S} w_s \leq w(MStT(G)) + \frac{1}{2} w(Alg)$ and hence $\frac{1}{2} w(Alg) \leq w(MStT(G))$. ■

This previous algorithm sorts the edges according to edge weights. Hence, if all the edges have the same weight, then any order will suffice.

Theorem 8. *There is a 2-approximate* **online stack** *priority algorithm solving the MStT problem when there are no edges between the Steiner nodes and the weight of every edge included is one.*

This upper bound together with the lower bound in Theorem 1 proves that the online stack model is incomparable with the adaptive priority model.

The above algorithms used the popping of the stack to get rid of the edges that are not in the required shortest paths. This is necessary. Davis and Impagliazzo in [6] shows that when each data item contains an edge, then though an adaptive priority algorithm without a stack can accept the shortest paths tree, (even when it is allowed to branch to a subexponential width), it cannot be designed to accept a shortest path from s to t (without accepting unneeded edges) [3].

Trivially, however, if shortest paths, i.e. the weights of the edges in G_R^c, are given as part of the input then the 2-approximation of the minimal Steiner tree problem is easy.

Theorem 9. *There is a 2-approximate* **fixed no-stack** *priority algorithm solving the MStT problem when the triangle inequality holds and the subgraph on the terminal nodes is complete.*

Proof of Theorem 9: By the triangle inequality assumption, the subgraph on the terminal nodes is G_R^c. Hence, finding a minimum spanning tree, $MST(G_R)$, of this subgraph gives a 2-approximation to $MStT(G)$. $MST(G_R)$ can be found using the standard greedy algorithm. Sort the terminal edges by weight and accept them if they don't form a cycle. ■

References

[1] M. Alekhnovich, A. Borodin, J. Buresh-Oppenheim, R. Impagliazzo, A. Magen, and T. Pitassi (2005) Toward a model for backtracking and dynamic programming, *IEEE Conference on Computational Complexity.*

[2] A. Borodin, D. Cashman, and A. Magen, (2005) How well can primal-dual and local-ratio algorithms perform?, *International Colloquium on Automata, Languages and Programming (ICALP).*

[3] J. Buresh-Oppenheim, S. Davis and R. Impagliazzo, (2011) A Stronger Model of Dynamic Programming Algorithms, *Algorithmica 60(4): 938 - 968.*

[4] A. Borodin, M. Nielsen, and C. Rackoff, (2002) (Incremental) Priority algorithms, *Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms.*

[5] D. Cashman, (2005) Approximate Truthful Mechanisms for the knapsack problem, and negative results using a stack model for local ratio algorithms, MSc Thesis, University of Toronto.

[6] S. Davis and R. Impagliazzo (2004) Models of greedy algorithms for graph problems, *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms.*

[7] M. R. Garey and D. S. Johnson (1979) *Computers and Intractability*, Freeman, San Francisco, CA.

[8] E. N. Gilbert and H. U. Pollak, (1968) Steiner minimal trees, *SIAM J. Appl. Math. 16 (1) 1-29*

[9] L. Kou, G. Markowsky and L. Berman, (1981) A Fast Algorithm for Steiner Trees, Acta Inf. 141-145.

[10] H. Kwon, (2008) Improved results on models of greedy and primal-dual algorithms, MSc Thesis, York University.

[11] K. Mehlhorn, (1988) A faster approximation algorithm for the Steiner problem in graphs, Information Processing Letters 27 (1988) 125-128.

[12] G. Robins and A. Zelikovsky (2000) Improved Steiner tree approximation in graphs, *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms.*

[13] H. Takahashi and A. Matsuyama (1980) An approximate solution for the Steiner problem in graphs, Math. Jap. 24 573-577

[14] V. Vazirani (2001) *Approximation Algorithms.* Springer.

[15] C. Gröpl, S. Hougardy, T. Nierhoff, and H. Prömel, Approximation Algorithms for the Steiner Tree Problems in Graphs, *Steiner Trees in Industries*, editors: D. Z. Du and X. Cheng, Kluwer.